Coopr Installation Guide 3.2

COLLABORATORS

	TITLE : Coopr Installation Guide 3.2		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	William E. Hart, Carl D. Laird, John Siirola, Jean-Paul Watson, and David L. Woodruff	October 11, 2013	

Contents

Preface

This book describes different methods for installing the Coopr software. Coopr is a collection of Python software packages that supports a diverse set of optimization capabilities for formulating and analyzing optimization models. A central component of Coopr is Pyomo, which supports the formulation and analysis of mathematical models for complex optimization applications. This capability is commonly associated with algebraic modeling languages (AMLs), which support the description and analysis of mathematical models with a high-level language. Although most AMLs are implemented in custom modeling languages, Pyomo's modeling objects are embedded within Python, a full-featured high-level programming language that contains a rich set of supporting libraries.

Goals of the Book

Unfortunately, Coopr is a complex software package that can be difficult to install. Coopr is comprised of a set of independent Python packages that need to be installed together. Coopr also depends on a variety of third-party Python packages, some of which have very different installation processes on different computer operating systems. Finally, Coopr can execute third-party optimization solvers, whose installation and configuration is completely independent of Coopr.

In this book, we outline a variety of installation options for different computer operating systems and for different usage models. These installation options reflect the different ways that Coopr developers and users have used Coopr, and they account for user access to system resources.

Comments and Questions

Further information about Pyomo and Coopr is available on the Coopr wiki:

`https://software.sandia.gov/trac/coopr`

Coopr is also hosted at COIN-OR:

`https://projects.coin-or.org/Coopr`

We strongly encourage feedback from readers about the software on the Coopr Forum:

`coopr-forum@googlegroups.com`

We hope this will include feedback on typos and errors in our examples in this book.

Good Luck!

Chapter 1

Overview

The Coopr download site includes Coopr installers for MS Windows and Unix platforms. Coopr is available open-source under the BSD license. It requires Python version 2.6 or 2.7. Coopr requires the installation of several freely available Python libraries, and it can optionally employ other libraries, including Python extension libraries for commercial optimization software.

There are two different ways that Coopr can be installed: (1) in the system Python directories, or (2) in a virtual Python that is installed in a user-specified directory. The former option is limited to users with administrative privileges. The latter option can be used by any user, but it does not create an installation that can easily be used by all users. The Installing in the System Python Environment chapter outlines different ways that Coopr can be installed in the system Python directories:

- Option 1: Use `easy_install` to install Coopr as a site package in your system Python installation
- Option 2: Use a MS Windows installer to install a Coopr release in your system Python installation
- Option 3: Use a MS Windows installer to install a version-of-the-day snapshot of Coopr in your system Python installation

The Installing in a Virtual Python Environment chapter outlines different ways that Coopr can be installed in a virtual Python environment:

- Option 4: Use the `coopr_install` script to install a Coopr release in a virtual Python installation
- Option 5: Use the `coopr_votd` script to install a version-of-the-day snapshot of the Coopr trunk in a virtual Python installation
- Option 6: Use the `coopr_votd` script to install the current Coopr trunk in a virtual Python installation

Additional installation details are provided in subsequent chapters:

- Prerequisites: Packages that you may need to install and run Coopr
- Installation Issues: Helpful information for resolving issues with Coopr installation.
- Platform Notes: Documentation of platform-specific issues, especially for installation.
- Solver Notes: Documentation for third-party solvers that can be called from Coopr.

Chapter 2

Installing in the System Python Environment

A standard way to install Python packages like Coopr is to install them as a Python site package. This generally requires administrative privileges, since Python is installed in system directories that are not accessible to general users.

2.1 Install a Coopr Release with the `easy_install` Script

Coopr can be installed with the following two steps:

1. Download and install the `setuptools` package

   ```
   wget http://peak.telecommunity.com/dist/ez_setup.py
   python ez_setup.py
   ```

2. Run `easy_install` to install Coopr:

   ```
   easy_install Coopr
   ```

The `easy_install` command downloads and installs Coopr from PyPI, which is the Python community's web-based service for hosting Python packages. This command handles package dependencies automatically, so there are no additional steps required to install Coopr.

Notes:

- This requires internet access to download Coopr and the packages it depends on.
- The first step can be skipped if `setuptools` is already installed.
- The `pip` package is now generally recommended over `setuptools`. If `pip` is installed, you can install Coopr with

  ```
  pip install Coopr
  ```

- You need to specify the `HTTP_PROXY` environment to install from PyPI through a proxy server.
- Coopr scripts are installed with other Python commands. You may need to add the directory for Python commands to your `PATH` environment to execute these scripts.

2.2 Install a Coopr Release with the MS Windows Installer

Warning
Coopr's MS Windows installer has not worked reliably for many users. We have recently updated this installer to address user issues. Please contact the Coopr developers if you have problems.

The MS Windows installer for Coopr installs each Coopr Python package as a Python site package. This does not require internet access; the MS Windows installer includes the Python packages that are needed to run Coopr.

MS Windows installer customizes the installation process as follows:

- The registry is checked to see if a previous Coopr installation needs to be uninstalled
- The user can optionally install additional Python packages that are used by Coopr (e.g. PyYAML and PyQt)
- A log of the install process is captured to help diagnose errors

Notes:

- Download the Coopr installer for the latest Coopr release: Coopr download site These installers have the name `Coopr_VERSION_DATE-setup.exe`.
- This package may need to executed with administrator privileges
- The user's environment is *not* adapted to add the Coopr binary directory to the user's path. The user must edit their `PATH` environment to include the `Scripts` directory in their Python installation. For example, Python 2.6 is normally installed in `C:\Python26\Scripts`. The `PATH` environment would need to be appended with `;C:\Python26\Scripts` for the Coopr executables to be executed from a DOS shell.

2.3 Install a version-of-the-day snapshot of Coopr with the MS Windows Installer

Warning
Coopr's MS Windows installer has not worked reliably for many users. We have recently updated this installer to address user issues. Please contact the Coopr developers if you have problems.

A version-of-the-day snapshot for Coopr can also be installed with a MS Windows installer. The installation process is the same as for the Coopr release installer. The version-of-the-day installer can be downloaded from the Coopr download site. These installers have the name `Coopr_VOTD-setup.exe`.

2.4 Getting Started

If Coopr is installed in the system Python environment, then the only configuration needed to use Coopr is setting up the `PATH` environment to include the system Python installation. A variety of Python scripts are included with Coopr. These scripts are available in the Python `bin` directory (Linux) or `Scripts` directory (MS Windows).

The simplest way to get started with Coopr is to import the `coopr` package that you wish to work with. For example, to use Pyomo, you would import `coopr.pyomo`:

```
\$ python
Python 2.5 (r25:51908, Nov  8 2006, 07:49:04)
[GCC 3.4.2 20041017 (Red Hat 3.4.2-6.fc3)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import coopr.pyomo
>>>
```

See Getting Started with Coopr for further discussion of how to use Coopr packages.

As a quick test of your installation, you can execute Coopr with the diet problem. If you installed Coopr with `easy_install`, then this example is available in the Python site packages directory, within `coopr.pyomo/examples/pyomo/diet`. If you installed Coopr with a MS Windows installer, then the this examples is available in the following directory:

```
YOUR_COOPR_INSTALL_DIR\examples\pyomo\diet
```

You can run the `pyomo` command to solve the diet problem:

```
pyomo diet1.py diet.dat
```

If glpk is installed on your system and the `glpsol` is available from the user `PATH` environment, then you should get an output like the following:

```
[    0.00] Setting up Pyomo environment
[    0.00] Applying Pyomo preprocessing actions
[    0.00] Creating model
[    0.03] Applying solver
[    0.05] Processing results
    Number of solutions: 1
    Solution Information
      Gap: 0.0
      Status: optimal
      Function Value: 2.81
    Solver results file: results.json
[    0.14] Applying Pyomo postprocessing actions
[    0.14] Pyomo Finished
```

Otherwise, Coopr will generate the following error message:

```
No executable found for solver 'glpk'
```

You can use the `--solver` option to specify an alternative solver, for example:

```
pyomo --solver=cbc diet1.py diet.dat
```

Chapter 3

Installing in a Virtual Python Environment

A virtual Python environment is an isolated Python installation that depends on the system Python installation, but which mimics a complete Python installation. A virtual Python environment can be created by a user in their directory without requiring administrative privileges. Thus, this installation option is available to all users.

3.1 Install a Coopr Release with the `coopr_install` Script

The `coopr_install` script installs Coopr in a virtual Python environment. Coopr can be installed with the following two steps:

1. Download the `coopr_install` script from the Coopr download site
2. Run `coopr_install` to install a Coopr release:

```
python coopr_install coopr
```

The `coopr_install` command installs Coopr from a cached version of this software that is integrated into the `coopr_install` script itself. Consequently, this installation process does not require internet access. Additionally, this command installs the associated packages that depends on, so there are no additional steps required to install Coopr. This script creates the virtual Python environment in the `coopr` directory, and the Coopr scripts are installed in the `coopr/bin` directory.

Notes:

- On MS Windows, this script cannot install all of the Python packages that Coopr depends on. Thus, this installer is not recommended for that platform.

3.2 Install a version-of-the-day snapshot of Coopr with the `coopr_votd` Script

A version-of-the-day snapshot for Coopr can also be installed with the `coopr_votd` script. The installation process is the same as for the Coopr release script, `coopr_install`. The `coopr_votd` script can be downloaded from the Coopr download site.

3.3 Install the current Coopr Trunk with the coopr_votd Script

The `coopr_votd` script also includes options that allow Coopr packages to be directly checked out from its subversion repository. These options are useful for advanced users who interact closely with the core developers, as well as users who wish to obtain updates for Coopr packages that are under very active development. This requires internet access to download Coopr and the packages it depends on, and it also requires the installation of the subversion software.

Coopr can be installed with the following two steps:

1. Download the `coopr_votd` package from the Coopr download site
2. Run `coopr_votd` to install Coopr. For example

   ```
   python coopr_votd --trunk coopr
   ```

   performs subversion checkouts for the the trunk branches of Coopr packages, which are then installed.

Although these options perform subversion checkouts, read-only access to the subversion repository does not require a password. The `coopr_votd` command handles package dependencies automatically, so there are no additional steps required to install Coopr. This script creates the virtual Python environment in the `coopr` directory, and the Coopr scripts are installed in the `coopr/bin` directory.

Notes:

- You need to specify the `HTTP_PROXY` environment to install from PyPI through a proxy server.
- The subversion software may need to be configured to work a proxy server on your network.
- The subversion software is commonly available on Linux and MacOS, but it is not preinstalled with MS Windows.

3.4 Getting Started

When Coopr is installed in a virtual Python environment, then the user needs to configure their `PATH` environment to include the system Python installation and the Coopr installation. A variety of Python scripts are included with Coopr. These scripts are available in the virtual Python `bin` directory, so the user needs to add the directory

```
YOUR_COOPR_INSTALL_DIR/bin
```

to the `PATH` environment. We recommend that the user add this directory **after** the directory for the system Python installation. This will ensure that the `python` command executes the system Python interpreter. The `coopr_python` command can be used to execute the virtual Python interpreter, which has Coopr installed.

The simplest way to get started with Coopr is to import the `coopr` package that you wish to work with. For example, to use Pyomo, you would import `coopr.pyomo`:

```
\$ coopr_python
Python 2.5 (r25:51908, Nov  8 2006, 07:49:04)
[GCC 3.4.2 20041017 (Red Hat 3.4.2-6.fc3)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import coopr.pyomo
>>>
```

See Getting Started with Coopr for further discussion of how to use Coopr packages.

As a quick test of your installation, you can execute Coopr with the diet problem. This example is available in the `YOUR_COOPR_INSTALL_DIR/examples/pyomo/diet` directory. You can run the `pyomo` command to solve the diet problem:

```
pyomo diet1.py diet.dat
```

If glpk is installed on your system and the `glpsol` is available from the user `PATH` environment, then you should get an output like the following:

```
[    0.00] Setting up Pyomo environment
[    0.00] Applying Pyomo preprocessing actions
[    0.00] Creating model
[    0.03] Applying solver
[    0.05] Processing results
    Number of solutions: 1
    Solution Information
      Gap: 0.0
```

```
      Status: optimal
      Function Value: 2.81
    Solver results file: results.json
[    0.14] Applying Pyomo postprocessing actions
[    0.14] Pyomo Finished
```

Otherwise, Coopr will generate the following error message:

```
No executable found for solver ’glpk’
```

You can use the `--solver` option to specify an alternative solver, for example:

```
pyomo --solver=cbc diet1.py diet.dat
```

Chapter 4

Prerequisites

4.1 Python 2.6 or 2.7

Coopr requires Python version 2.6 or 2.7. In particular, some components rely on Python modules that exhibited known bugs in Python 2.4. Further, we rely on and benefit from several packages and performance improvements introduced in Python 2.6. The Python language underwent some rather large changes to version 3.0. These changes are not backward-compatible with Python 2.x, and consequently Coopr does not currently not work with Python 3.0.

- Windows users:

  The Python web site provides binary installation downloads for Windows and Mac. Recall that the PATH environment variable may need to be updated, as the installers typically do not update it automatically. On Windows, this can be easily accomplished via the Control Panel "Advanced System Settings" panel.

- Linux/UNIX users:

  Most flavors of Unix ship with Python, and a compatible version of Python is typically and easily installed using your distribution's package manager. Note that you will need to install both the `python` package and (if separate) the `python-devel` package (to get the Python header files that are needed to compile some external applications distributed with Coopr). Most Linux installers put the Python executable on a common location on your path. If this is not the case for your particular distribution, you can add the location to your shell's PATH environment variable.

- Mac users:

  The Python web site provides binary installation downloads for Mac. Recall that the PATH environment variable may need to be updated, as the installers typically do not update it automatically.

Please be aware of 32-bit versus 64-bit issues, particularly if you are running Windows. In particular, if running a 64-bit Windows OS (likely at this point, but not so with XP or earlier versions), you should use the Python 2.X.Y Windows X86-64 installer, as opposed to the (32-bit) Python 2.X.Y Windows installer. If you run 32-bit Python on a 64-bit Windows platform, for example, you won't have access to any more than 3GB (actually a bit less) of RAM - even though the machine probably has 8GB or more.

Note that several Python distributions are available that are bundled with convenient packages (e.g. SciPy and MatPlotLib):

- Python(x,y)
- ActiveState
- Enthought

4.2 Subversion

It may be necessary to install to have subversion installed on your system to execute the `coopr_install` installation script. On Unix systems, this is commonly pre-installed. On Windows, free installer packages can be obtained from CollabNet.

Only the command-line client is necessary; specifically, the "CollabNet Subversion Command Line" distribution should be used. Do *'not'* install "CollabNet Subversion Edge", as it downloads and installs Python 2.5. While the CollabNet download is free, CollabNet will require you to create a login - presumably to track usage statistics. The installer updates the system PATH variable automatically.

4.3 Pywin32

On Windows, it may also be necessary to install the Python Windows extensions. This package enables Python to support Windows COM interfaces, and specifically it allows Pyomo to work with Excel spreadsheets. Note that "pywin32" is a bit of a misnomer, in that it works fine on 64-bit platforms; based on what we can tell, the "32" just means "non-16-bit".

The installer for this extension is available from http://sourceforge.net/projects/pywin32. This install simply updates the python site packages.

4.4 PyODBC

Coopr requires the installation of PyODBC to access ODBC data sources. On Windows, the PyODBC package can be installed by downloading a Windows installer, or by executing `easy_install`:

```
easy_install pyodbc
```

On Unix-based systems, (including Linux and Mac OS X) will require several system-level packages for proper ODBC functionality. These include:

- `unixODBC`: provides generic ODBC connectivity
- `unixODBC-devel`: includes ODBC development headers for linking with other libraries
- `mdbtools`: implements Microsoft Jet (Access) database interaction
- `libmdbodbc`: bridges Jet databases via ODBC

Finally, users on such Unix-based systems may have to configure their local ODBC installations for Jet databases if using Microsoft Access. To do so, add the following to the file `/etc/odbcinst.ini`:

```
[Microsoft Access Driver (*.mdb)]
Description = MDB Tools ODBC drivers
Driver      = /usr/lib/libmdbodbc.so.0
Setup       =
FileUsage   = 2
```

Pyomo will attempt to add any further ODBC information as needed.

Chapter 5

Installation Issues

Here are some of the most common issues encountered by Coopr users. If you do not see your question answered here, the best route for getting an answer is by engaging the Coopr community through the coopr-forum mailing list.

5.1 Python Installation

The standard Python distribution is built with the C language, and thus this is sometimes called CPython. CPython distributions are available for all major platforms, and this is the most common version of Python that is used in practice. Linux and Macintosh distributions commonly include CPython, and no additional configuration is required to execute the Python interpreter.

On Windows platforms, CPython can be easily installed with an installer executable. However, the user must edit the PATH environment variable to include the Python installation path.

Other noteworthy implementations of Python are Jython, written in Java, IronPython, written for the Common Language Runtime, and PyPy, a Python interpreter writen in Python. A major difference between CPython, Jython and IronPython is that they support extensions with C, Java and .NET application framework respectively. PyPy supports these extensions depending on the underlying Python interpreter, and it uses a JIT compiler to improve the execution of Python code.

Coopr is developed and tested using CPython. Other Python implementations have been evaluated, and their status is summarized here:

- Jython: The `virtualenv` package is not supported in Jython. This package is critical for the development, testing and deployment of Coopr.
- IronPython: This Python implementation does not currently support installation with `setuptools` or `distribute`, which are commonly used installation mechanisms in CPython. Coopr depends on these packages.
- PyPy: Many Coopr package install with PyPy. There remain some issues with reference counting in Python, which are documented in Coopr ticket #4437.

Most Coopr packages are designed to work with version 2.6 and 2.7. The Python language underwent some rather large changes in version 3.0. These changes are not directly backward-compatible with Python 2.x, and consequently Coopr does not currently work with Python 3.x.

5.2 Coopr Installation

5.2.1 Help with `coopr_install`

There are numerous options to the `coopr_install` script, which can be displayed by typing:

```
./coopr_install --help
```

Important
Make sure you do *'not'* have the `PYTHONHOME` environment variable defined prior to installing Coopr. Such a definition interferes with the construction of the Coopr virtualized Python environment.

5.2.2 Using Preinstalled Site Packages

By default, the 'coopr_install` command does *'not'* mirror the contents of the system Python installation site-packages directory. This helps ensure the stability of the Coopr installation. However, there are situations where a user may want access to pre-installed site-packages, like SciPy and/or NumPy. To enable this behavior, simply specify the "--site-packages" option at installation.

Another option is to install these packages directly into the virtualized Python environment created by `coopr_install`. However, installing Python packages can be tricky, particularly if they rely on compiled extension packages.

5.2.3 Using the `HTTP_PROXY` Environment Variable

In many computing environments, it may be necessary to set the `HTTP_PROXY` environment variable to use the `wget` and `coopr_install` commands. A typical value for this variable is something like "http://foo.institution.domain:80". Your local system administrator can help you assess whether you need an HTTP proxy for web access. For example, at Sandia (New Mexico only) the proxy `http://sonproxy.sandia.gov:80` is used for the SON network, and `http://wwwproxy.sandia.gov:80` is used for the SRN network.

5.2.4 Installing with Package XXX

The `-a` or `--add-package` option can be used to install Coopr with a specified package. That this option can specify a directory that contains a package that needs to be installed. Alternatively, this option can specify a package name that is downloaded from PyPI. Note that package installation may fail for complex packages like SciPy that use compiled Python extensions. For these cases, the package needs to be installed separately, or a Python bundle like Python(x,y) needs to be used.

5.2.5 Installing on MS Windows

There is a known problem with MS Windows Coopr installations. Coopr depends on the PyYAML package, which does not install under MS Windows with the 64-bit Python build. Thus, Coopr requires the 32-bit build for MS Windows.

5.2.6 Installation error: Filename ... does not start with any of these prefixes: ...

We have seen the error:

```
Installation error: Filename ... does not start with any of these prefixes: ...]
```

when installing on Windows. This appears to be a limitation of the virtual environment logic. Specifically, this was triggered by explicitly specifying the Python path with a lower-case drive name. For example:

```
c:\Python27\python.exe coopr_install
```

Using a capital drive name resolved this issue:

```
C:\Python27\python.exe coopr_install
```

5.2.7 Installation error: Setup script exited with error: command *gcc* failed . . .

Several Python packages that Coopr relies on (e.g., coverage and PyYAML) include compiled "C" extensions. This error indicated that the extensions did not successfully compile. In most cases, the root cause is that the Python C headers are not installed or available on your system. For Linux/UNIX users, make sure you have the `python-devel` package installed.

5.3 Using Subversion with a Proxy

Subversion does not use the `HTTP_PROXY` environment variable to configure its behavior. Rather, this must be done by modifying the local subversion configuration. As expected, this differs on Windows and Unix platforms, and is environment-specific. However, the step-by-step process is somewhat generic. Sandia-specific instructions can be found [sandbox:UsingSubversion here]. The same instructions should be applicable in other environments by appropriately substituting the name of your local proxy server.

5.4 Python environment variables

The `PYTHONPATH` and `PYTHONHOME` environment variables are used to customize the way that Python is configured. The `PYTHONHOME` variable defines the location of the standard Python libraries that are used, and `PYTHONPATH` augments the default search path for module files.

Since Coopr is installed in a virtual Python environment, these environment variables are generally not necessary. We have seen many cases where Coopr scripts failed to operate properly when these variables were defines, so we generally recommend that the user disable these variables before installing Coopr and executing Coopr scripts.

Chapter 6

Platform Notes

We have successfully installed and executed Coopr on a wide range of platforms. In the course of doing so (and helping other users do so), we have identified a number of issues that are worth mentioning, and easily correctable.

6.1 Windows XP, Vista, and 7

An issue we have observed on Windows Vista is the following error:

```
WindowsError: [Error 740] The requested operation requires elevation
```

This issue is due to the new and "improved" Windows Vista permissions scheme. Basically, `python.exe` needs to run with administrator privileges, e.g., to compile things in site packages. To accomplish this, simply navigate via Explorer to where `python.exe` is located (typically `C:\Python2.6`) and right-click on the `python.exe` icon. Then, select the "Compatability" tab and check the "Run this program as an administrator" box.

In addition, some distributions of CPLEX and AMPL appear to have overloaded the name for the `CPLEX.exe` executable. Coopr currently interacts with CPLEX via the command-line, interactive version, and not the version with the AMPL/NL driver. The symptom of this situation is a failed solve, which can be easily diagnosed by turning on the "--keepfiles" option in pyomo and looking at the CPLEX output log. The fix is to simply make sure the interactive CPLEX is first in your PATH environment variable.

6.2 Windows 7

To install on Windows 7 (and possibly other Windows systems), it is necessary to run with administrator privileges when installing Coopr. Otherwise, the `coopr_install` script will generate error messages indicating the lack of write permission in various directories. If installing from the command prompt, one can right-click on the "Command Prompt" icon and select "Run as administrator" to obtain such permissions. It is worth noting that this is not always necessary, as in many environments users are automatically granted administrator privileges.

Various users have observed issues when running `coopr_install` under Windows 7, related to the subversion client obtained from http://tortoisesvn.tigris.org. In particular, you may observe the following error:

```
Can’t move ’.svn/tmp/entries’ to ’.svn/entries’: The file or directory is corrupted and ↩
    unreadable.
```

To quote the Tigris site:

```
This error message typically occurs when you try to update or commit
your working copy, and seems to be common on Windows 7 systems. It
is due to another process holding a handle on a file that Subversion
```

```
needs to move or modify. This might be a virus scanner, but on
Windows 7 it is likely to be the Windows Indexing Service. Turn off
the indexing service on your working copies and repositories, and
exclude them from virus scans.
```

Some combination of disabling the indexer and the antivirus software seems to correct the problem; please let us know if this is not the case.

6.3 Red Hat Linux

No issues involving installation on Red Hat systems have been observed.

6.4 Ubuntu Linux

Some distributions of Ubuntu do not install by default with subversion. Users and/or their system administrators will need to install this package.

Similarly, some Ubunti distributions do not appear to have a full python 2.6 install. Specifically, the `cProfile` package can be missing, and will need to be installed.

6.5 MacOS (Snow Leopard)

At least one user has reported problems during execution of the `coopr_install` script on MacOS, specifically related to the `easy_install` of the "psutils" package. When attempting to compile this package, one may observe numerous compile errors relating to various system header files. This issue appears to be due to ARCHFLAGS on some systems defaulting to the PowerPC architecture. To fix this issue, first execute:

```
sudo ARCHFLAGS="-arch i386 -arch x86_64" easy_install psutil
```

Next, re-execute `coopr_install`. With `psutil` successfully installed, `coopr_install` will simply use the installed package and skip the re-install.

Chapter 7

Solver Notes

Coopr includes interfaces to a variety of third-party solvers. We do not attempt to document the installation of these packages. This chapter provides notes for how the user environment needs to be configured so Coopr can automatically detect and execute these solvers.

7.1 ASL Solvers

The AMPL Solver Library (ASL) provides a general interface for reading an AMPL NL file. Coopr provides a general interface for any solver that employs that ASL. That is, Coopr executes ASL solvers with the same command-line syntax that is used by AMPL, and the SOL result files are read to represent the solver results in Coopr. Coopr can execute NL files generated by AMPL or Coopr's Pyomo modeling package.

See the AMPL web pages for a summary of the ASL solvers that are available. Many of these have commercial support, and there are a variety of mature open source ASL solvers.

7.2 CBC

CBC is an open-source solver for linear programs and mixed-integer linear programs. CBC is written in C+\+, and it provides a stand-alone executable.

See the CBC wiki for download and installation instructions. The directory containing the `cbc` executable must be in the list of paths defined by the `PATH` environment variable.

7.3 CPLEX

The IBM ILOG CPLEX Optimizer includes commercial mathematical programming solvers for linear programming, mixed integer programming, quadratic programming, and quadratically constrained programming problems.

Coopr has solver interfaces for the CPLEX command as well as the CPLEX Python environment. To use the CPLEX command, the directory containing the `cplex` executable must be in the list of paths defined by the PATH environment variable. There are a variety of ways that the CPLEX Python environment can be used with Coopr. Perhaps the simplest is to specify the CPLEX Python directory when installing Coopr with the `coopr_install`. For example, if the CPLEX Python directory is `/usr/local/ilog/cplex/python`, then you can install Coopr with this package installed as follows:

```
coopr_install --add-package=/usr/local/ilog/cplex/python coopr
```

7.4 GLPK

The GLPK (GNU Linear Programming Kit) package is intended for solving large-scale linear programming (LP), mixed integer programming (MIP), and other related problems. It is a set of routines written in ANSI C and organized in the form of a callable library.

See the GLPK website for download and installation instructions. The directory containing the `glpsol` executable must be in the list of paths defined by the PATH environment variable.

7.5 GUROBI

The Gurobi Optimizer is a commercial solver for linear programming (LP), quadratic programming (QP) and mixed-integer programming (MILP and MIQP).

Although Gurobi Optimizer supports a Python environment, Coopr only interfaces with the Gurobi command. The directory containing the `gurobi.sh` (Linux) or `gurobi.bat` (MS Windows) executable must be in the list of paths defined by the PATH environment variable.

7.6 PICO

PICO is a solver for linear programming and mixed-integer linear programming problems that can perform parallel optimization on distributed memory machines.

PICO is a component of Acro, an open-source software project that integrates a variety of optimization software packages, including both libraries developed at Sandia National Laboratories as well as publicly available third-party libraries. Acro's Getting Started wiki pages provides instructions for downloading and installing Acro projects that include PICO. The directory containing the `PICO` executable must be in the list of paths defined by the PATH environment variable; typically this will be the `acro/bin` directory.

Colophon

This book was created using `asciidoc` software.

The Sandia National Laboratories Laboratory-Directed Research and Development Program and the U.S. Department of Energy Office of Science Advanced Scientific Computing Research Program funded portions of this work.

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. SAND 2012-1572P.